A 2's COMPLEMENT DIGITAL MULTIPLIER THE Am2505

By R. C. Ghest, Digital Applications

ADVANCED MICRO DEVICES

APPLICATION NOTE
NOVEMBER 1971

6, AVENUE CH. DE GAULLE, HALL
78150 LE CHESNAY
TÉL. 954.91.13

INTRODUCTION

This paper describes an MSI multiplier that can be used as an element in an iterative array to perform 2's complement multiplication at high speed without using large amounts of hardware. The device allows direct hardware multiplication to be performed in minicomputers and also permits Fast Fourier Transform processors and time-shared digital filter networks to be designed quickly and economically.

At present, multiplication in digital machines is usually performed sequentially by the processor arithmetic logic unit. Sequential methods of this sort, however, are time-consuming, and any attempt to speed up the computation by special-purpose control and shifting logic substantially increases the amount of hardware.

BASIC IDEAS IN MULTIPLICATION

Multiplication consists of a series of single-bit multiplications and parallel additions. A decision is made either to add the multiplicand to the partial product or to add zero, depending upon whether the multiplier digit under consideration is a '1' or a '0'. After each addition, the partial product is shifted one place relative to the multiplicand, and the next multiplier digit is examined for the succeeding iteration.

Multiplication can proceed by examining the least significant bit of the multiplier first and proceeding to the most significant, or by examining the most significant bit first and proceeding to the least significant. As a rule, examination starts from the least significant end, in which case only a single-length adder is required. If examination starts from the most significant end, carry signals have to propagate the full length of the partial product and a double-length adder is required.

Multiplication is performed in the following manner:

$$\text{Let } X = x_0 2^0 + x_1 2^1 + x_2 2^2 + x_3 2^3$$

$$\text{Let } Y = y_0 2^0 + y_1 2^1 + y_2 2^2 + y_3 2^3$$

$$X \cdot Y = (x_0 2^0 + x_1 2^1 + x_2 2^2 + x_3 2^3) \times (y_0 2^0 + y_1 2^1 + y_2 2^2 + y_3 2^3)$$

If $y_i = 0$, then '0' is added to the partial product.
If $y_i = 1$, the multiplicand X is added in to the partial product shifted by the weight associated with the multiplier digit under consideration.

$$\begin{aligned} X \cdot Y = {} & y_0 2^0 (x_0 2^0 + x_1 2^1 + x_2 2^2 + x_3 2^3) \\ & + y_1 2^1 (x_0 2^0 + x_1 2^1 + x_2 2^2 + x_3 2^3) \\ & + y_2 2^2 (x_0 2^0 + x_1 2^1 + x_2 2^2 + x_3 2^3) \\ & + y_3 2^3 (x_0 2^0 + x_1 2^1 + x_1 2^1 + x_2 2^2 + x_3 2^3) \end{aligned}$$

It can be seen from the preceeding example that the time required for multiplication is n-1 times the time required for a parallel addition where n is the number of bits in the multiplier.

HIGH-SPEED MULTIPLICATION

Various schemes for reducing multiplication time are presently being used. The most obvious is to ignore additions when the multiplier bit under consideration is '0'. When this is done the number of additions is equal to the number of '1's in the multiplier. This method can be extended in such a way that strings of '1's in the multiplier can also be ignored, which leads to an important new technique for performing high-speed multiplications. This technique will be discussed in greater detail.

HIGH-SPEED MULTIPLICATION USING CARRY-SAVE ADDERS

During multiplication, the time taken for the various additions is determined to a large extent by the generated carries propagating through the system. This carry time can be reduced by not allowing carries to propagate at the same level, but by adding them at the next level. With this technique, multiplication can be performed at high speed, but (1) a great deal of hardware is required, and (2) at the end of the array a parallel adder has to be used to add the final sums and carries, and the carries must propagate the entire length of the adder.

HIGH-SPEED MULTIPLICATION USING ROMs

If a small high-speed multiplier can be built, it can be used with other identical circuits and adders to build larger multipliers. High-speed ROMs can be used to build a small multiplier with no carries, and the various partial products can be added together using high-speed adders. This technique has four major disadvantages: (1) the ROM has to be used with its associated custom masks; (2) large amounts of hardware are necessary; (3) an iterative type structure is not possible; and (4) it does not perform 2's complement arithmetic directly.

HIGH-SPEED MULTIPLICATION USING SQUARING ARRAYS

It has been suggested that multiplication be performed by making use of one of the following formulas:

$$2AB = (A + B)^2 - A^2 - B^2$$

$$4AB = (A + B)^2 - (A - B)^2$$

A squaring array requires considerably less hardware than a multiplying array. When a squaring array is used, the operands A, B must first be added and then the square or squares subtracted. Therefore, the total amount of hardware required to do multiplication using squaring arrays is considerably greater than that required using iterative structures, and the squaring array gives no increase in speed.

EXAMINATION OF SEVERAL MULTIPLIER BITS

With the usual method of digital multiplication, the multiplier digits are examined in turn, and when a digit is a '1', the multiplicand is added to a running total or partial product in the appropriate weight. For each multiplier digit, there is a relative shift between multiplicand and partial product, whether there has been an addition or not. If, however, shifting is not limited to one stage and several multiplier bits are examined at a time, the process may be speeded up.

For example, the product of the words 1 0 1 1 0 1 1 x 1 1 1 1 1 1 1 (most significant bit at the right-hand end) requires seven shifts and additions by the normal method. However the multiplier may be expressed in a different form:

$$1\,1\,1\,1\,1\,1\,1 = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6$$

which may be written $1\,1\,1\,1\,1\,1\,1 = -2^0 + 2^7$ or $\bar{1}\,0\,0\,0\,0\,0\,0\,\overset{+}{1}$ where the signs have their usual arithmetic meaning.

If multiple shifting is allowed with the introduction of subtraction, the product may be evaluated with a single shift of seven places. In general, the numbers involved will not be simple sequences of '1's or '0's but may consist of alternating strings.

Thus: $1\,1\,1\,0\,0\,0\,0\,1\,0\,1\,1\,1\,0\,0\,1\,0$

$= \bar{1}\,0\,0\,\overset{+}{1}\,0\,0\,0\,\bar{1}\,\overset{+}{1}\,1\,0\,0\,\bar{1}\,\overset{+}{0}\,\bar{1}\,\overset{+}{1}$

$= \bar{1}\,0\,0\,\overset{+}{1}\,0\,0\,0\,\overset{+}{1}\,\bar{0}\,\overset{+}{1}\,0\,0\,\bar{1}\,\overset{+}{0}\,\overset{+}{1}\,0$

CONSTANT SHIFT OF TWO PLACES

If the number of places to be shifted is made constant, the shifting logic is eliminated, and an iterative array structure is possible since a constant shift can be made by suitable connection.

A set of rules that governs the arithmetic operation can be derived from the previous examples. Note that the present operation depends not only on the two multiplier bits under consideration but also on whether the previous operation was an add or a subtract. If the shift is always two places, then the multiplier bits must be grouped in pairs, and combinations of signs can occur only in the two bits under consideration. Hence the above example becomes

$11,\bar{1}0,00,\overset{+}{0}1,\bar{0}1,\overset{+}{1}1,00,\bar{1}0$

$= \bar{1}0,\overset{+}{0}1,00,\bar{0}1,\overset{+}{1}1,00,\bar{1}0,\overset{+}{1}1$

$= \bar{1}0,\overset{+}{0}1,00,\bar{0}1,\overset{+}{1}0,00,\bar{1}0,\overset{+}{1}0$

and the table is

y_i	y_{i+1}	Previous Operation	Present Operation
0	0	Add	Do nothing.
1	0	Add	Add multiplicand.
0	1	Add	Subtract 2 x multiplicand.
1	1	Add	Subtract multiplicand.
0	0	Subtract	Add multiplicand.
1	0	Subtract	Add 2 x multiplicand.
0	1	Subtract	Subtract multiplicand.
1	1	Subtract	Do nothing.

It can be seen that instead of storing the fact that an add or subtract has occurred, all that is necessary is to examine the previous bit y_{i-1} of the original sequence.

$\bar{1}1,\overset{+}{1}0,\overset{+}{0}0,\bar{0}1,\bar{0}1,\bar{1}1,\overset{+}{0}0,\overset{+}{1}0$

$\bar{1}0,\overset{+}{0}1,00,\bar{0}1,\overset{+}{1}0,00,\bar{1}0,\overset{+}{1}0$

A new table can now be drawn up.

y_{i-1}	y_i	y_{i+1}	Operation
0	0	0	Add zero.
1	0	0	Add multiplicand.
0	1	0	Add multiplicand.
1	1	0	Add 2 x multiplicand.
0	0	1	Subtract 2 x multiplicand.
1	0	1	Subtract multiplicand.
0	1	1	Subtract multiplicand.
1	1	1	Subtract zero.

From the table it can be seen that:

The y_{i+1} bit can be used as an add/subtract indicator.

The function $y_{i-1} \oplus y_i$ indicates that 1x the multiplicand should be added or subtracted to the partial product.

The function $y_{i-1}\, y_i\, \bar{y}_{i+1} + \bar{y}_{i-1}\, \bar{y}_i\, y_{i+1}$ indicates that 2x the multiplicand should be added or subtracted to the partial product.

The algorithm generated above has several important features:

It produces directly, without correction, a 2's complement product from a multiplier and multiplicand represented in 2's complement notation. If the multiplier's most significant digit is a '1', a subtraction must take place at the last bit pair since y_{i+1} for this pair is always '1'. For a 2's complement, an addition at the n^{th} weight is not required, because 2's comp (y) = $2^n - y$.

The algorithm allows multiplication in either the active HIGH (positive logic) or active LOW (negative logic) representations simply by changing when an addition or subtraction takes place. An adder/subtractor performs the same operations in either logic representation, and the only multiplier decoder function that changes with change of logic polarity is y_{i+1} to $\bar{y}_{i+1}$.

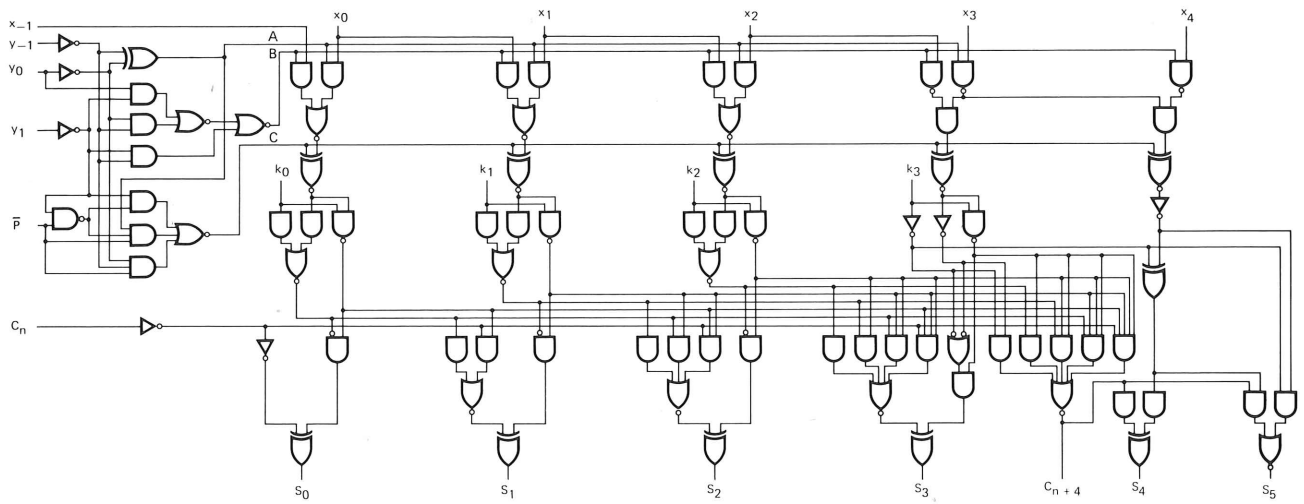
Multiplier bits can be scanned from the most to the least significant digit. When this is done, additional multipliers are required in the array because carries must propagate the complete length of the partial product. Although this method does allow a double length number to be added to the product, more efficient methods of doing this are available.

AN ITERATIVE MSI MULTIPLIER

The Am2505 is an MSI element that obeys the multiplication algorithm developed previously. It can be used to multiply signed or unsigned numbers in various number representations and performs multiplications in either positive or negative logic.

The logic diagram of the device is shown in Figure 1, the logic symbols and pin numbers in Figure 2. The circuit consists of five parts: a multiplier decoder, a shifting array, a complementer, a high-speed adder, and an overflow and sign control.

LOGIC DIAGRAM



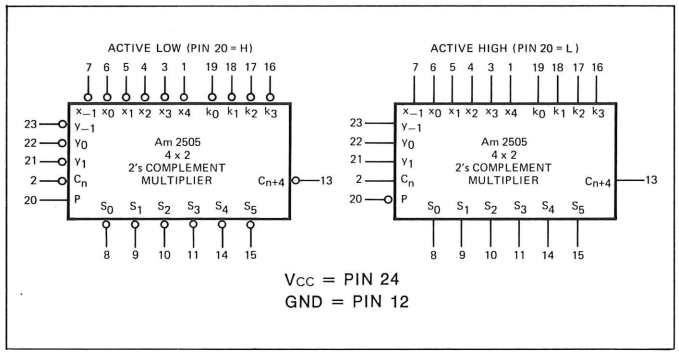


Figure 2. Am2505 4 x 2 Signed Multiplier Logic Representations

1) Multiplier Decoder
This decoder generates three control signals which indicate whether 1x or 2x the multiplicand is to be used and whether an add or subtract is required. The decoder generates the functions

$$A = y_{i-1} \oplus y_i \quad \text{(1 x multiplicand)}$$

$$B = y_{i-1}\, y_i\, \bar{y}_{i+1} + \bar{y}_{i-1}\, \bar{y}_i\, y_{i+1} \quad \text{(2 x multiplicand)}$$

$$C = \bar{P}\, \bar{y}_{i+1} + P(y_{i+1}\, A + \bar{y}_{i-1}\, \bar{y}_i) \quad \text{(add or subtract)}$$

The input P controls the add or subtract sequence so that the multiplier can work in either the positive or negative logic representation. The function is complicated by the shifting array giving HIGH outputs for logic '0' independent of the logic representation.

2) Shifting Array
The shifting array sends either 1x or 2x the multiplicand or all HIGH's to the complementer.

3) Complementer
This consists of a set of exclusive OR circuits controlled by the add/subtract control.

4) High-Speed Adder
This part consists of a 4-bit high-speed parallel carry look-ahead adder that adds the operated multiplicand to the partial product presented at the K inputs. The adder has a carry input and a carry output and four sum outputs, S_0 to S_3.

5) Overflow and Sign Control
At the most significant end of the array, a problem arises when an overflow occurs as a result of (a) an addition or subtraction or (b) the need to use 2x the multiplicand. To overcome these overflow situations, the sign digits of the multiplicand and partial product must be repeated twice. Luckily some logic minimization is possible, and the two outputs, S_4 and S_5, which are the most significant bits of the 6-bit signed product from a 2-bit signed multiplier and 4-bit signed multiplicand can be generated quite easily. These two outputs are required only at the most significant end of each iterative step of a multiplication. In order to reduce input loading on x_3 an additional x_4 input is provided which must be connected to x_3 at the most significant end of the array.

ITERATIVE ARRAYS USING Am2505s

Since the Am2505 is a 2x4 multiplier and performs the function

$$S = XY + K,$$

it can be used as a large iterative cell in multiplication schemes. The number of multiplier devices required is given by $\frac{M \times N}{8}$ where M and N are the word lengths of multiplier and multiplicand.

Figure 3 shows four Am2505s connected to form a 4 x 8 array that produces a 2's complement product from a 4-bit 2's complement multiplier and an 8-bit 2's complement multiplicand. The scheme is shown for the positive logic representation; for the negative logic representation, P must be left open rather than held low to form a logic LOW, and '1's and '0's must be reinterpreted. Since the first iteration is treated as if the previous operation were an addition, the x-1, y-1 inputs are held at logic '0'. The S_4, S_5 outputs are ignored except at the most significant edge of the array. The K inputs allow the accumulation of partial products as information passes through the array. Since at the first stage the partial product does not exist, the K inputs can be used to add in a number at the least significant end of the product. This feature is very useful as many arithmetic processes consist of a series of multiplication and additions. For multiplication with longer word lengths, the array can be extended in both the X and Y directions as described below.

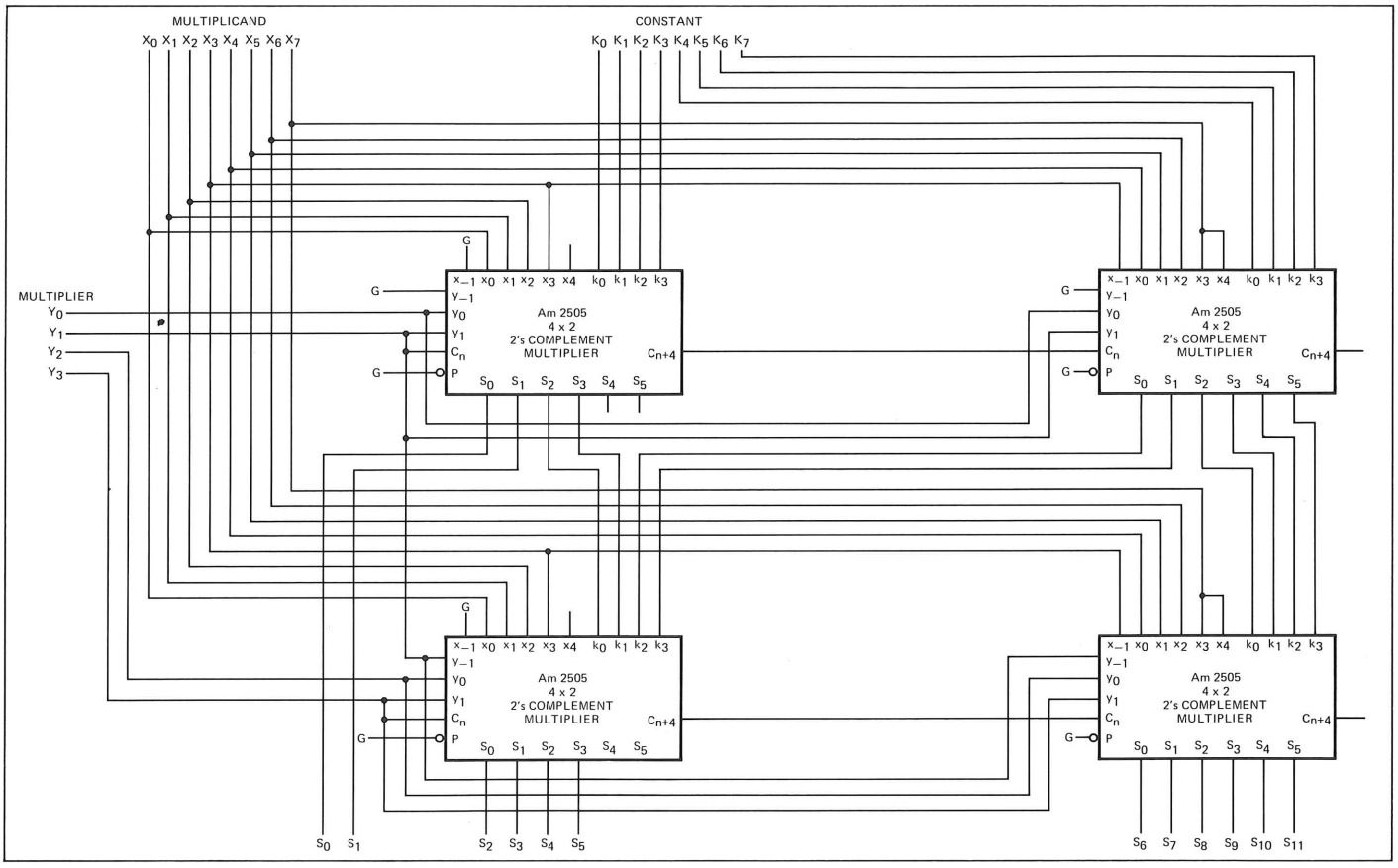


Figure 3. 2's Complement 8 x 4 Multiplication. Active High Levels

Figure 4 shows the straightforward method of stacking multipliers so as to accumulate partial products and generate a product. This array, which consists of 2-bit multipliers staggered by 2-bit positions in a parallelogram-type structure, is not the fastest possible. Its speed, however, can be increased by physically moving some multiplier blocks, all the while making sure that the relative weight of the partial product is not changed. This means that each multiplier can change its relative vertical position in the array but not its horizontal position.

Figure 5 diagrammatically shows different connection schemes for a 16 x 16 bit multiplier with the fastest array possible shown as method 3. In these diagrams 64 represents the multiplier with $Y_6 = y_5\ y_6\ y_7$ and $X_4 = x_3\ x_4\ x_5\ x_6\ x_7$ as inputs. When the multipliers are moved in the array, all the x and y inputs must stay with the multiplier, and only the k and internal carry array lines are broken. Multipliers on the right hand edge of the array cannot be moved because they carry sign information.

The connection between y_1 and C_n inputs at the left side of the array can be broken to free the C_n input for another signal, but the y_1 line must then be connected to another carry input with the same weight as the original one.

If faster multiplers are required the multiplication array can be split into several parts and the partial products from these parts added using high-speed full look-ahead adders such as the Am9340. This method, shown in Figure 6, results in a substantial increase in speed—especially for large multipliers—with relatively few packages.

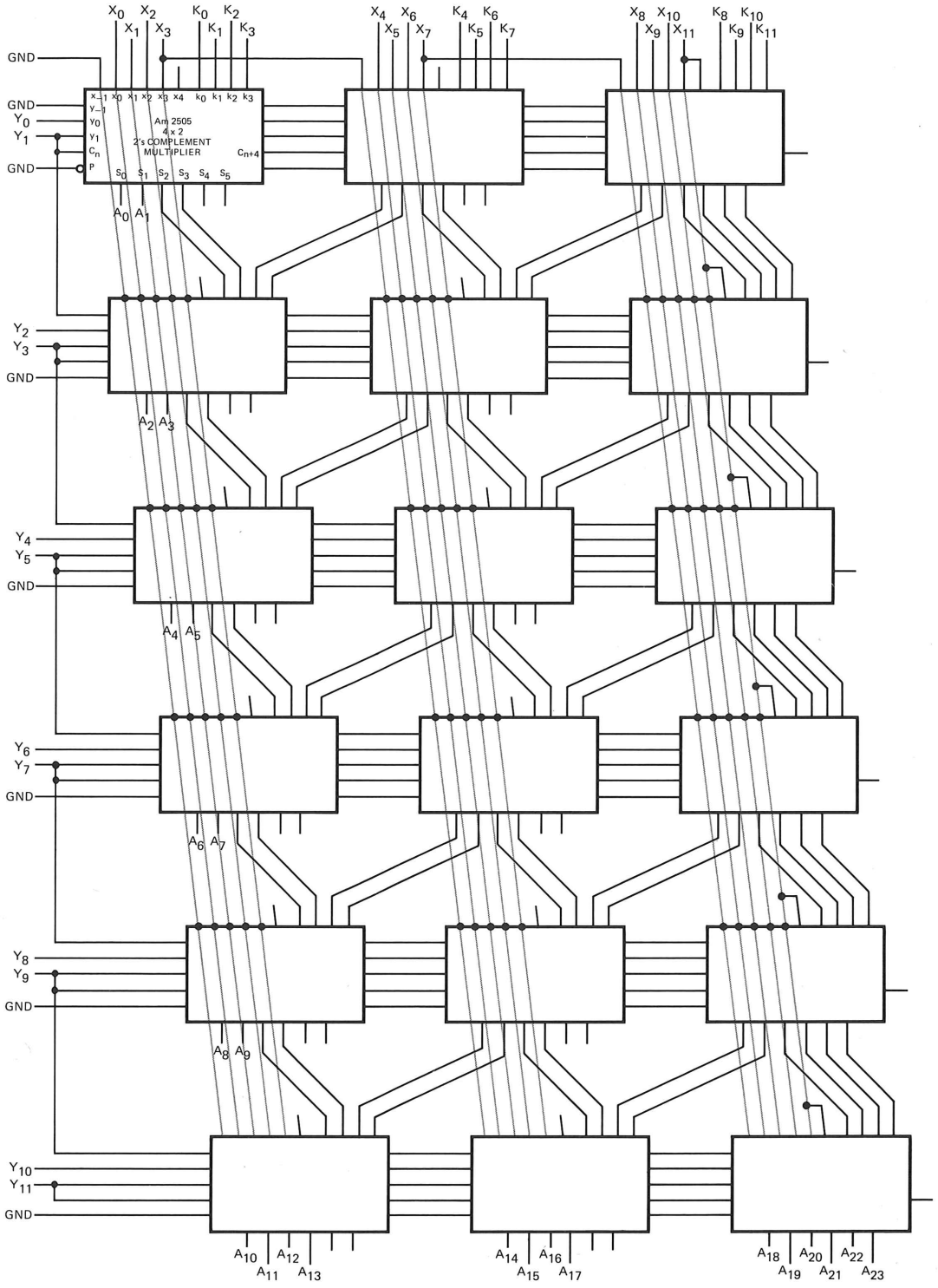
This method of splitting the array into parts can be used to further decrease the average multiplication time by storing the partial products in latches, then adding the stored parts with high speed adders, and storing the final product; the Am2506 Arithmetic Logic Unit with output latch is ideal for this purpose. The time to complete the first multiplication in a series of multiplications is increased but the average time is decreased to the delay of an array part and the delay of the latch used to store the partial product.

Table I shows the delays and package count for various size multipliers. Where no additional packages are used, the array uses Method 3, the optimum speed method. A 12 x 12 multiplier using this connection scheme is shown in Figure 7.

TIME-SEQUENCED MULTIPLIERS

The Am2505 can be used as the main element in a time-sequenced multiplier. This is illustrated in Figure 8. The multiplier and partial product are shifted two places after each iteration. Three single-length registers are required: one holds the multiplicand; the other two hold the double-length product. The least significant part of this double-length register originally holds the multiplier, which is gradually shifted out during the computation. A shift of two places is obtained by splitting the multiplier and partial product up into odd and even parts and placing the odd bits in one shift register and the even bits in the other. A shift of one place of both registers then effectively acts as a shift of two places.

The scheme can be extended to use any number of even multiplier bits. As the number of bits increases, the multiplication time increases, and the amount of hardware increases. When Am2505s are used in a combinational array, the array does not require many devices, and time-sequenced multipliers are worthwhile only if the word lengths are long or if the auxiliary registers can be shared with other arithmetic operations.



(Method 1) INTEGERS FRACTIONS

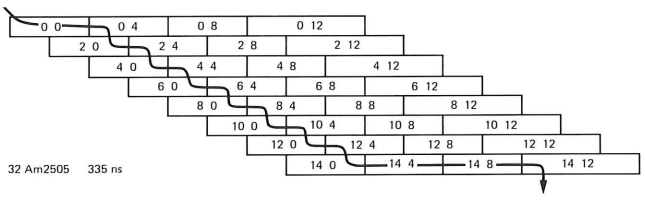
Figure 4. 12 x 12 Signed multiplier $A = (X \cdot Y + K)$ or $A = X \cdot Y + K2^{-12}$

TABLE I

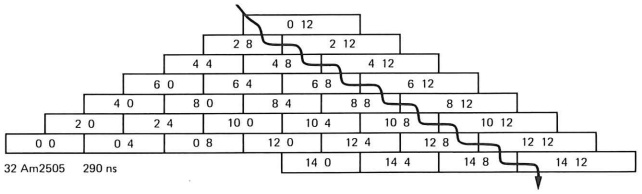
DELAYS AND PACKAGE COUNT

Array Size	Number of Am2505s for 2n–bit Product	Number of Additional Am9340s	Typical Delay Times in ns at 25°C Level of Adders Am9340		
			0	1	2
8 x 8	8	0	135		
12 x 12	18	0	205		
	18	5		155	
16 x 16	32	0	275		
	32	7		185	
	32	16			180
20 x 20	50	0	345		
	50	9		220	
24 x 24	72	0	415		
	72	11		255	
	72	24			215
28 x 28	98	0	485		
	98	13		290	
32 x 32	128	0	555		
	128	15		325	
	128	32			250

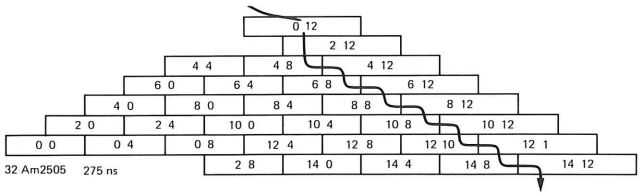
Maximum delays at 25°C add 40% of typical delay.



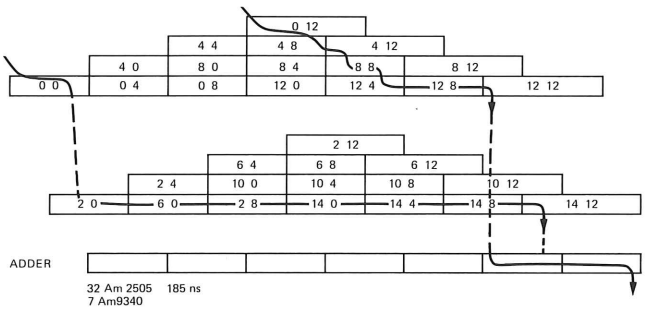
(Method 1)



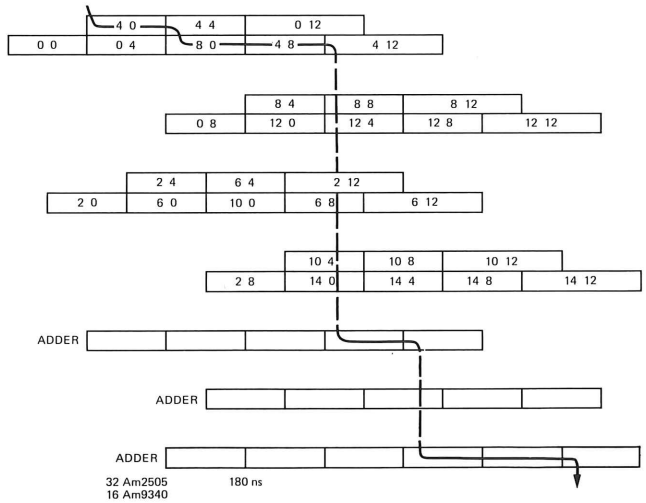
(Method 2)



(Method 3)



(Method 4)



(Method 5)

Figure 5. Multiplier connection schemes.

Figure 6. Multiplier connection schemes with additional adders.

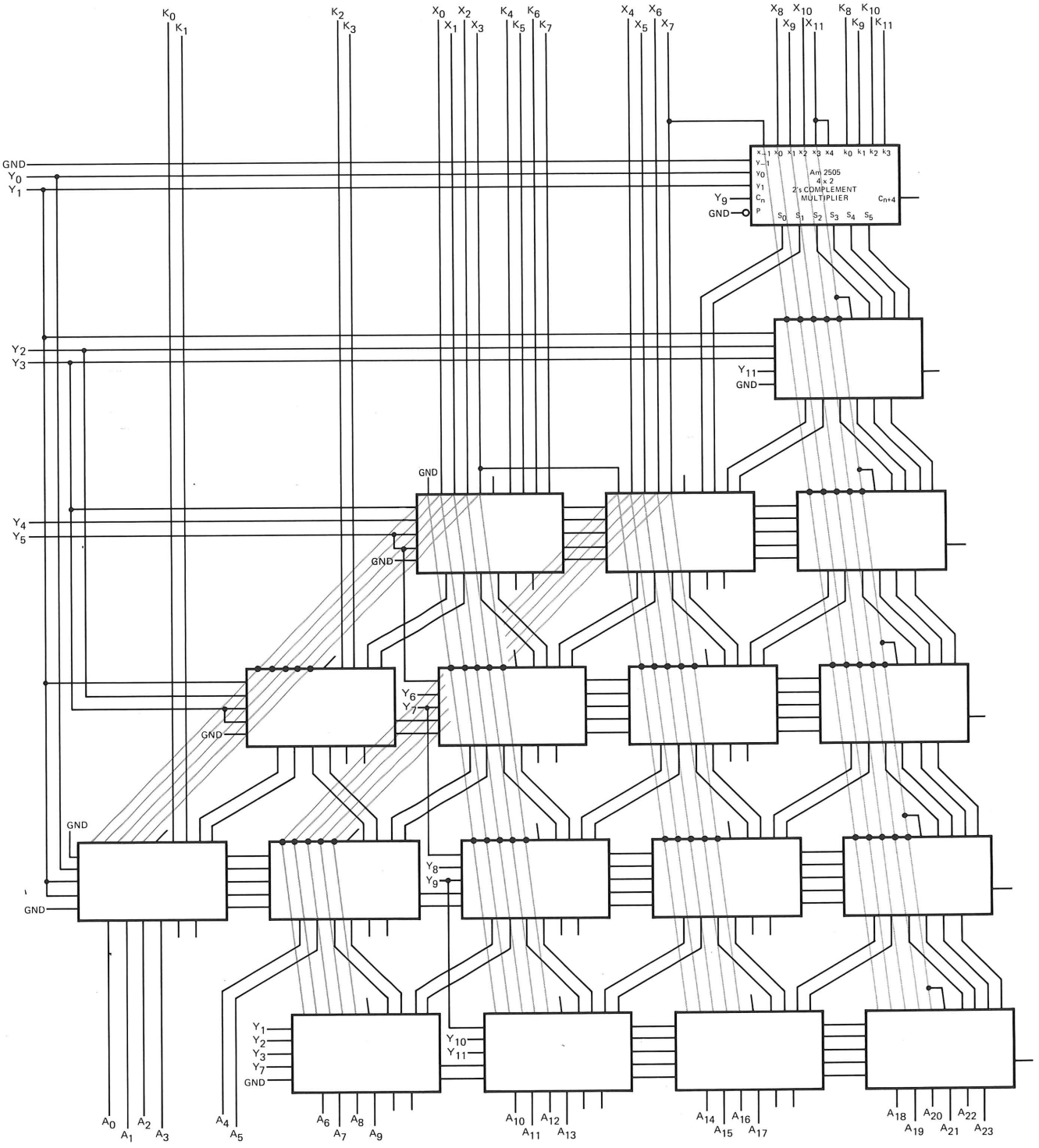


Figure 7. 12 x 12 Signed multiplier A = (X · Y + K) (Method 3)

ROUND-OFF

It is often convenient to use only the most significant half of a product. This product should be rounded off; that is, it should approximate the best n-bit answer possible. This can be done by examining the least significant half of the product, and if it is greater than or equal to a certain value, (normally 1/2 that of the least significant digit of the truncated product) adding a '1' to its most significant position.

Forming a rounded n-bit product from a conventional product constrained within the range $-1 \leqslant S \leqslant 1\text{-}2^{-(2n-2)}$ can be done by adding a '1' to the K input at weight 2^{-n}, that is, one k position lower than the K sign digit. An example of rounding off is shown below.

X =	0.	0	1	1				= 3/8
Y =	0.	1	0	1				= 5/8
XY =	00.	0	0	1	1	1	1	= 15/64
+K =	00.	0	0	0	1	0	0	
S =	00.	0	1	0	0	1	1	

Rounded n-bit product from 2n-bit product

S =	0.	0	1	0	= 1/4

TRUNCATION

If the user is prepared to accept a truncated product where the product is incorrect by at most the least significant digit, the number of packages necessary for multiplication can be decreased. Each 4 x 2 multiplier contributes at most 31 times some binary weight to the final answer. For example in a 12 x 12 multiplication to form an n-bit truncated product, the first multiplier in row 0 contributes a maximum of 31×2^0, the next multiplier in the same row contributes a maximum of 30×2^4, the first multiplier in the next row contributes a maximum of 31×2^2, etc. Altogether, five multipliers can be removed from the 12 x 12 array and the product still kept accurate to within plus or minus the least significant digit in the truncated n-bit product. When products are truncated, no speed increase occurs since the removed multipliers are not in the only critical speed path.

Table II gives the package count for double-length untruncated 2n-bit and n-bit truncated square multiplier arrays.

TABLE II

Array Size	Number of Packages: 2n-bit Product	Number of Packages: Truncated n-bit Product
8 x 8	8	6
12 x 12	18	13
16 x 16	32	22
20 x 20	50	33
24 x 24	72	46
28 x 28	98	61
32 x 32	128	78

MULTIPLICATION IN OTHER NUMBER REPRESENTATIONS

Although 2's complement multiplication is the one most widely used, multiplication in other number representations often has to be performed. The Am2505 can be used to generate a 2's complement product which is then corrected in an appropriate manner.

UNSIGNED (MAGNITUDE) MULTIPLICATION

Extend the multiplier at the least significant end one bit to give

$$(X)_2 = x_o + x - x_s 2^n \qquad x = x_1, x_2 \ldots .$$
$$(Y)_2 = y_o + y - y_s 2^n \qquad y = y_1, y_2, y_3 \ldots .$$

Let $x_s = y_s = 0$ Then $(XY)_2 = (XY)_m$

and $(XY)_m = xy + (x_o y + y_o x) + x_o y_o$

A conditional adder can be used to generate $x_o y + y_o x + x_o y_o$ and the result added to the product xy at the K inputs. The first two stages of the conditional adder produce the first two bits of the product. The remaining product digits are produced at the output of the multiplier. The sign digits x_s, y_s and k_s are held at logic 0 and the two most significant multiplier bits are ignored. Note that since the correction is performed at the K inputs the delay is the same as for a 2's complement multiplication.

SIGN-MAGNITUDE MULTIPLICATION

$$(X)_{sm} = x(1-2x_s)$$
$$(Y)_{sm} = y(1-2y_s)$$
$$(XY)_{sm} = xy(1-2x_s)(1-2y_s)$$
$$(XY)_2 = xy - 2^{n-1}(x_s y + y_s x) + 2^{2n-2} x_s y_s$$
$$(XY)_{sm} = (1-2x_s)(1-2y_s)[(XY)_2 + 2^{n-1}(x_s y + y_s x) - 2^{2n-2} x_s y_s]$$

There are four conditions for $x_s y_s$:

$x_s y_s$	$(XY)_{sm}$
0 0	$(XY)_2$
1 0	$-[(XY)_2 + 2^{n-1} y]$
0 1	$-[(XY)_2 + 2^{n-1} x]$
1 1	$[(XY)_2 + 2^{n-1}(x + y) - 2^{2n-2}]$

Since the items to be added or subtracted are at a weight n-1 or higher, they must operate on the most significant part of the product. The item $-2^{2n-2} x_s y_s$ is generated by adding in 1 at weights 2n-2 and 2n-1 when $x_s y_s = 1$.

1's COMPLEMENT MULTIPLICATION

$$(X)_1 = x - x_s(2^{n-1} - 1)$$
$$(Y)_1 = y - y_s(2^{n-1} - 1)$$
$$(XY)_1 = xy - 2^{n-1}(x_s y + y_s x) + (x_s y + y_s x) + x_s y_s (2^{n-1} - 1)^2$$
$$(XY)_2 = xy - 2^{n-1}(x_s y + y_s x) + x_s y_s 2^{2n-2}$$
$$(XY)_1 = (XY)_2 + x_s Y_2 + y_s X_2 + x_s y_s$$

There are four conditions for $x_s y_s$:

$x_s y_s$	$(XY)_1$
0 0	$(XY)_2$
1 0	$(XY)_2 + (Y)_2$
0 1	$(XY)_2 + (X)_2$
1 1	$(XY)_2 + (X + Y)_2 + 1$

Since the items to be added to the 2's complement product are at weight 0, the K inputs can be conveniently used for this purpose. A conditional adder can be used to produce the function $x_s (Y)_2 + y_s (X)_2 + x_s y_s$, and the sum can be added to the multiplier array at the K inputs.

APPLICATIONS

The multiplier is ideal for hardware multiplication in general and special purpose computers. However, its main application at present would appear to be in digital filter circuits, FFT processors, and arithmetic function generators. In the applications shown in Figures 10 - 13, the multiplier array is presented as a box which performs the function

$$S = XY + K$$

and care must be exercised in scaling the numbers appropriately.

Figure 10 shows how two multiplier arrays can be connected to generate a quadratic in x. This can be extended to form polynomials with higher powers of x.

The multiplier can also be used in an iterative time-sequenced mode where a register is used to hold partial results. Figure 11 shows the function $S = \frac{1}{1-x}$, which if y = 1-x, is the reciprocal of y.

A faster method to find the reciprocal of a number is to use two multiplier arrays and Newton's approximation method as shown in Figure 12. This can obviously be extended to solve equations.

Figure 13 shows how the square root of a number can be found by comparing the number with x^2, where x is contained in a counter. The counter is initially at zero and incremented for each iteration.

A change of sign of S indicates that $x^2 \geqslant P$ and that the computation should stop. If a faster method is required, x can be generated not from a counter but from a successive approximation register, such as the type used in A/D converters.

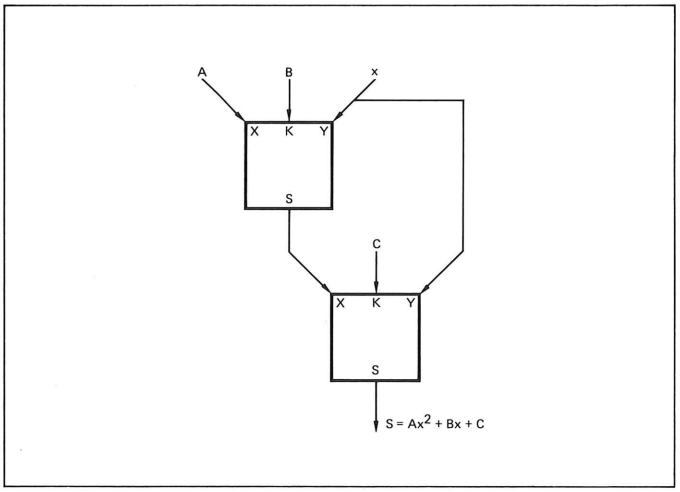


Figure 10. Polynomial evaluation

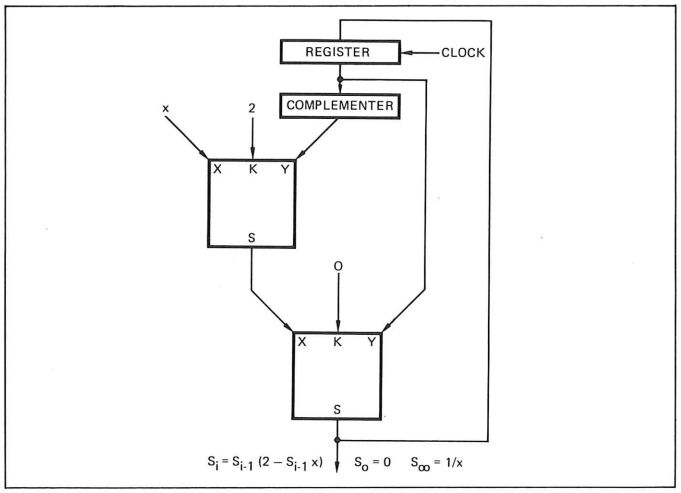


Figure 12. Reciprocal evaluation.

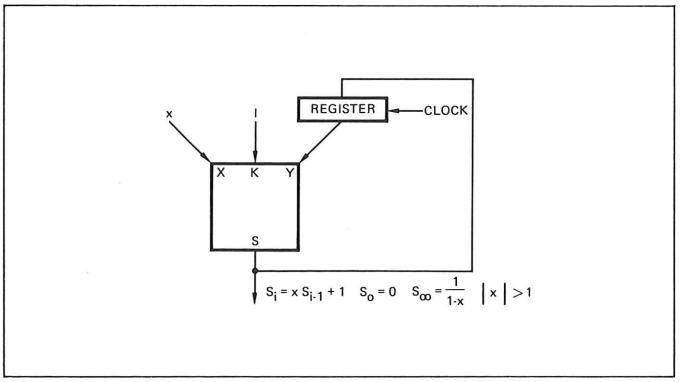


Figure 11. Reciprocal evaluation.

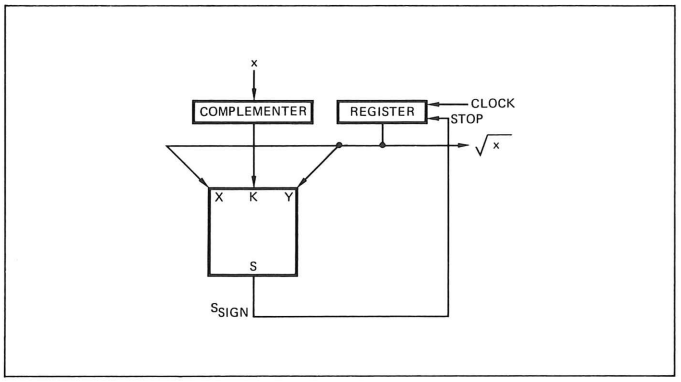
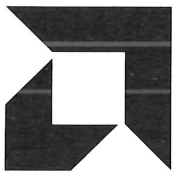


Figure 13. Square root generation.



ADVANCED
MICRO
DEVICES INC.
901 Thompson Place
Sunnyvale
California 94086
(408) 732-2400
TWX: 910-339-9280
TELEX: 34-6306